Charon-Suite Module Framework
Modular Algorithms with Serializable C++ Objects

Jens-Malte Gottfried    Daniel Kondermann

Heidelberg Collaboratory for Image Processing (HCI)
Interdisciplinary Center for Scientific Computing (IWR)
University of Heidelberg

IPOL 2012 Meeting on Image Processing Libraries
Outline

1. Charon-Suite
   - Serializable Objects
   - Workflows

2. Helper Tools
   - Tuchulcha
   - Template-Generator
   - Examples and Documentation
   - Precompiled Binaries

3. Application
   - Optical Flow Estimation
   - 3D Reconstruction
   - Parallel Calculations
Project Overview
What is Charon-Suite? Why was it created?

- image processing libraries difficult to maintain and extend in a research environment, steep learning curve, short life-cycle
- Charon-Suite is a framework with associated tools rather than a library

Charon-Suite
- open source framework for computer vision prototyping
- independent of any given image processing library
- simple plugin-architecture for parts of computer vision algorithms
- modules may use any language and any software package
- graphical helper tools for configuration and execution
- easy to learn, cross-platform
Used Software
Dependencies and Build Tools, License Information

**framework**
- framework written in C++
- build using cmake
- documentation with doxygen
- GUI elements using Qt4
- core and helper classes under GNU LGPLv3
- platform independent; supported: Win32/64 with MsVC, Linux/GCC, Mac/GCC

**modules**
- arbitrary libraries may be used: CImg, Vigra, Qt, Petsc, OpenCV and more
- wrapper modules for other languages; already available: python, matlab, scripts
- most modules also use LGPL, but other licenses possible

---

Charon-Suite Module Framework

J.M. Gottfried, D. Kondermann (HCI)
Charon Modules
Dividing Algorithms into Parts using Serializable Objects

- Algorithmic steps divided into different modules (e.g. read data, processing steps, write result)
- Modules encapsulate algorithm parameters and their documentation
- Load/store parameters from/to parameter files (plain text)
- White-box testing of parts and full algorithms possible
- Re-usability of existing modules without introducing new bugs
Module Interaction
Data Flow/Slot Model

- slot mechanism for data exchange between modules
- full algorithm described by interaction of modules
- directed acyclic graph required

- save connections also in the plain text parameter file
- flow chart visualization
- execution by traversing the modules of this graph
## Tool Overview

Simplify Usage and Development

### main tools

<table>
<thead>
<tr>
<th>Tool</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>templategenerator</td>
<td>set up code templates for new modules</td>
</tr>
<tr>
<td>tuchulcha</td>
<td>graphical workflow configuration and execution tool</td>
</tr>
<tr>
<td>paraminspector</td>
<td>parameter file editor, standalone object inspector</td>
</tr>
</tbody>
</table>

### tools for scripting

<table>
<thead>
<tr>
<th>Tool</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>workflow-executor</td>
<td>command line workflow executor</td>
</tr>
<tr>
<td>charon-xml-helper</td>
<td>check module groups in doxygen documentation</td>
</tr>
</tbody>
</table>

### web-based services

<table>
<thead>
<tr>
<th>Tool</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sourceforge</td>
<td>code hosting, wiki, tutorials, project web, win releases</td>
</tr>
<tr>
<td>launchpad</td>
<td>code hosting, ubuntu package repository, recipe-builds</td>
</tr>
</tbody>
</table>
Tuchulcha
Graphical Workflow Configuration

- overview of available modules
- module documentation
- graphical workflow and connection management (like LabView or similar)
- parameter adjustment using the object inspector
- workflow execution
- qt event loop for display modules
- node/workflow comments
- parameter priorities
Template Generator
Code Templates for New Modules

- wizard simplify generating new modules
- specification of class name, input/output slots and parameters with their types and documentation
- generation of C++ files `module.h` `module.hxx` `module.cpp` and CMake `CMakeLists.txt`
- placeholder to add execution code
The project page at sourceforge is the place to start working with the Charon-Suite framework. There is information for beginners as well as for experienced users.

- project wiki with tutorials, build instructions and links to other information sources
  http://charon-suite.sf.net
- doxygen documentation
  http://charon-suite.sf.net/doc/project
- tool references, usage and options (manpages)
  http://charon-suite.sf.net/doc/man
- example workflows with detailed description and needed data
  http://charon-suite.sf.net/doc/examples
Testing
Test Suite and Dashboard Overview

Charon-Suite Testing
- test suite for base classes and module collections
- automated tests based on CTest
- continuous and nightly tests (using scripts)
- automated documentation, examples and manpage generation
- test result overview using CDash dashboard
  http://charon-suite.sf.net/CDash/
Precompiled Binaries

Using Charon-Suite without compiling anything

- get Charon-Suite running in a few minutes only
- download and unpack, run tuchulcha
- use example workflows with demo algorithms
- adapt these workflows to your needs

available binaries

- Win32/Win64 MsVC 2010
- Ubuntu Linux DEB Packages
  Repository with all dependencies
- Gentoo Linux Ebuilds
  (layman overlay)

Module Collections
Overview of Existing Modules

charon-utils  data input/output, image manipulation
charon-flow  optical flow estimation and related modules
hekate     3D reconstruction and related modules
charon-petsc parallel calculations using MPI

**charon-utils**: supported data formats

- cimg, pgm, bmp
- jpeg, tiff, png via external libraries
- all formats supported by ImageMagic
- hdf5 via vigra library
# Charon-Flow

Various Optical Flow Estimation Algorithms

## Working Algorithms

- Horn and Schunck (1981)
- Lukas and Kanade (1981)
- Combined local/global (Bruhn et al. 2005)
- Nonlinear and multiscale versions (Pyramids)
- Learning flow (Sun et al. 2008)
- Charbonnier functions (Charbonnier et al. 1997) (Papenberg et al. 2006)
- Classic+NL (Sun et al. 2010)
- Range-Flow (Scene-Flow)

## Helpers

- CImgList based iterators (loops in workflows)
- Contrastive divergence (CD) learning algorithm (Hinton 2002)
- Monte Carlo sampling (Metropolis et al. 1953)
- PDE solvers based on Petsc, conjugate gradients, SOR, Cuda Version (approx.), 2nd Order Newton
Hekate
Camera Calibration, Feature Detection and 3D Reconstruction

Algorithms

- 3D Reconstruction by Structure from Motion (SfM)\textsuperscript{12}
- Feature Detection and Tracking (for example SIFT features)
- Outlier elimination of feature correspondences\textsuperscript{23}
- Delaunay triangulation to generate meshes
- Auto-calibration of cameras by image sequence
- Results can be used for camera tracking

References


For more information ask Moritz Becker
Charon-Petsc
MPI-Based Workflows

features

- multi processing using `mpirun`
- wrapping parallel vector and sparse matrix classes
- export to matlab vectors
- operations like add and multiply with matrices and vectors
- parallel filtering (derivation, convolution with gaussian)
- converters from and to existing image types (VigraMultiArray)

work-in-progress

- wrapping KSP solver class
- mixing parallel and non-parallel modules

For more information ask Gerald Mwangi
Conclusion

summary
- open source framework
- modular architecture
- stable since 2009
- increasing number of available algorithms
- precompiled binaries for fast setup
- no need to reinvent the wheel
- application not restricted to computer vision
	hanks to
- Daniel Kondermann (project initiator)
- Stephan Meister (charon-utils, argos)
- Michael Baron (charon-flow)
- Moritz Becker (hekate)
- Gerald Mwangi (charon-petsc)